ECET SENIOR DESIGN PROJECT REPORT

Indianapolis Motor Speedway (IMS) Display Project

Submitted to

Professor Robert Weissbach
Professor William Lin
Electrical Engineering Technology Program
Engineering Technology Department
by
Charleston Shi and Cristobal Ibanez

December 10, 2019

Document Information

Title: IMS Display Qualification Testing
Author: Charleston Shi and Cristobal Ibanez
File Name: FINAL REPORT

Revision History

Date	Initials	Version	Section Updated	Notes
9/17/19	C.S	1.0	Document Created	
8/1/19	C.S	2.0	Updated signatures	
11/15/19	C.S	3.0	FINAL REPORT	Pre-Final Presentation
12/5	C.S	4.0	FINAL REPORT	Post-Final Presentation

Table of Contents

Definitions

The following definitions, acronyms, and abbreviations are found in this document:

Table 1 - Definitions

Term	Definition
CAN Bus	Standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer
IMS	Indianapolis Motor Speedway; referring to the museum
PLC	Programmable Logic Controls
LCD	Liquid Crystal Display; referring to the display screen on the steering wheel
Xtrac	Referring to the sponsor
IndyCar	Premier level of open-wheel racing in North America

Abstract

The Indianapolis Motor Speedway Museum and Xtrac want an interactive display of a racecar transmission and allows people of all ages to witness learn. Xtrac has commissioned IUPUI engineers to create a control box that correlates to other functions of a racecar and then correspond said box to a steering wheel, with additional features specified by the sponsor. Since a transmission has mostly a mechanical aspect, Mechanical Engineering Technology (MET) students are paired with a team of EET/CPET students. Any mounting and specification requirements are a part of the MET students' project requirements. Details regarding electrical power, circuit design, and electromechanical integration will be generated by the EET students.

Keywords: ECET, Python, Xtrac, PLC, code, museum, transmission box, microcontroller, steering wheel, testing, IMS, IndyCar, Captstone

Introduction

The Indianapolis Motor Speedway (IMS) Museum has a display transmission that is still actively used in IndyCar today. The museum desires to create an interactive display that incorporates this transmission so visitors of all ages can see internal gears spin as well as see the transmission shift between its gears. The transmission will have a user-friendly interface of an actual racecar steering wheel that will send signals to and fro the transmission, allowing museum guests of all ages to shift gears, change speed, and turn the transmission on and off. The outcome of this project is a failsafe and robust system that will operate within specifications set by the IMS Museum while being continuously updated.

Xtrac has commissioned IUPUI engineers to create a museum-quality interactive simulation of a racecar to be put on exhibit for the Indianapolis Motor Speedway Museum. For this project, we will be primarily focused on corresponding a racecar steering wheel to the control box, with additional features specified by the sponsor. Both steering wheel and engine control box are materials that have been donated for our project. It should be noted that we are also inheriting projects from previous engineer teams, and thus are trying to simultaneously reconcile various design choices in order to better the already existing project. By the end of the Fall 2019 semester, we are expecting the project to be completed and be placed on display at the IMS Museum.

The overall objective of the IMS Project is to develop or redesign an interface of a racecar steering wheel with an engine control box. The resulting connection should send signals back and forth between steering wheel and engine. Critical information should be displayed on the screen and the LED lights. Certain aspects of the scope will include integrating the steering wheel with the control box utilizing CAN Bus. The expected supply voltage and supply current for the steering wheel is 10-18 V and 230 mA, respectively. You will shift gears using 6 steering wheel buttons and/or 2 paddle and display information on the steering wheel (using screen controls to shift gears and LED indicators to determine when to shift gears). This process will be done by connecting the information from the gearbox and sending it through a Raspberry Pi 3B to do necessary calculations then converting the information into CAN bus for the steering wheel to read.

Specification Requirements

The objective of the IMS Project is to develop or redesign programmed interface of the donated steering wheel in order to interface with a control box that manipulates a motor display and displays critical information by the end of Fall 2019 semester. Certain aspects of the scope will include:

- Integrating the steering wheel with the control box utilizing CAN Bus. The expected supply voltage and supply current for the steering wheel is 10-18 V and 230 mA, respectively.
- Shift gears using 6 steering wheel buttons and/or 2 paddles.
- Display information on the steering wheel
    - Using screen controls to shift gears
    - LED indicators to determine when to shift gears

Out of Scope

The following items are outside the scope of this effort:

- Design of the screen on the steering wheel
- Identification/color of buttons

As of 2019, the current engineering technology design students have since inherited the project left behind by the previous electrical design group.
The task will now be to correct the actuator that influences the gearbox before interfacing with the microcontroller. This task is not considered

Assumptions

The final setup will be as a simulated display of a racecar engine on a museum that will see communication between a steering wheel and a gearbox transmission. Regular maintenance and supervision will be done.

Gentle usage of the display per museum rules to prolong the life of the display is suggested for all operators and users.

Related Projects

Mechanical engineer technology design students are in charge of three main points: redesigning/modifying the linkage between the actuator and the gear shift, designing the safety box, and the steering wheel mount.
Another group of electrical engineer technology design students are in charge with the entirety of the project and are in the process of designing ladder logic in connected components workbench primarily.

User Roles and Responsibilities

Table 2 - User roles and responsibilities

Role	Responsibility
Programmer	Responsible for researching and programming code
Researcher	Compile and compose all data for documentation and presentation
Tester	Take measurements and data from experimentations

Process Overview

Current Process (System)

The display is separated into three projects. It begins with the engine, within the engine a device will be added to shift the gears mechanically. Next a control box will be added to the display to control the engine's gears with the added device as well as take information from the engine i.e (rpm, temperature, current gear shifted). Lastly, the part of the project currently being address is to take the information from the control box and display it to a donated steering wheel from Cosworth. This steering wheel will also send information to the control box to change gears on the engine and control the rpm.

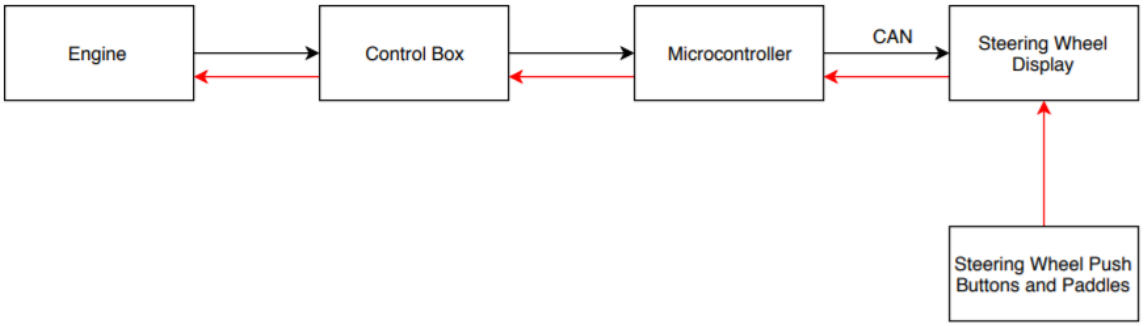
This process will be done by connecting the information from the gearbox and sending it through a Raspberry Pi 3B to do necessary calculations then converting the information into CAN bus for the steering wheel to read. To shift gears, 2 pins will be used to output signals from the paddle shifters to the Raspberry Pi or directly to the control box if possible to then shift the gear up or down.

High-Level Design

System Components

As represented in Figure 1, the engine will send signals through the PLC in the control box, which will direct said signals to a microcontroller. Through CAN communications, the microcontroller interfaces with the steering wheel and display any measurements analyzed from the signals on the LCD screen and LEDs. Interacting with the push buttons and the paddles on said steering wheel will send signals to the steering wheel display and sent signals back through the microcontroller with CAN communications to the control box to the engine.

Figure 1: Hardware Flow Diagram



Concept of Execution

As displayed in Figure 2, when the engine starts, the LCD display will showcase elements (RPM, speed, gear shift, etc...). Through event handlers and interrupts, the program will send data to the control box and update the LCD display per the button chosen.

Figure 2 – Software Flow Diagram

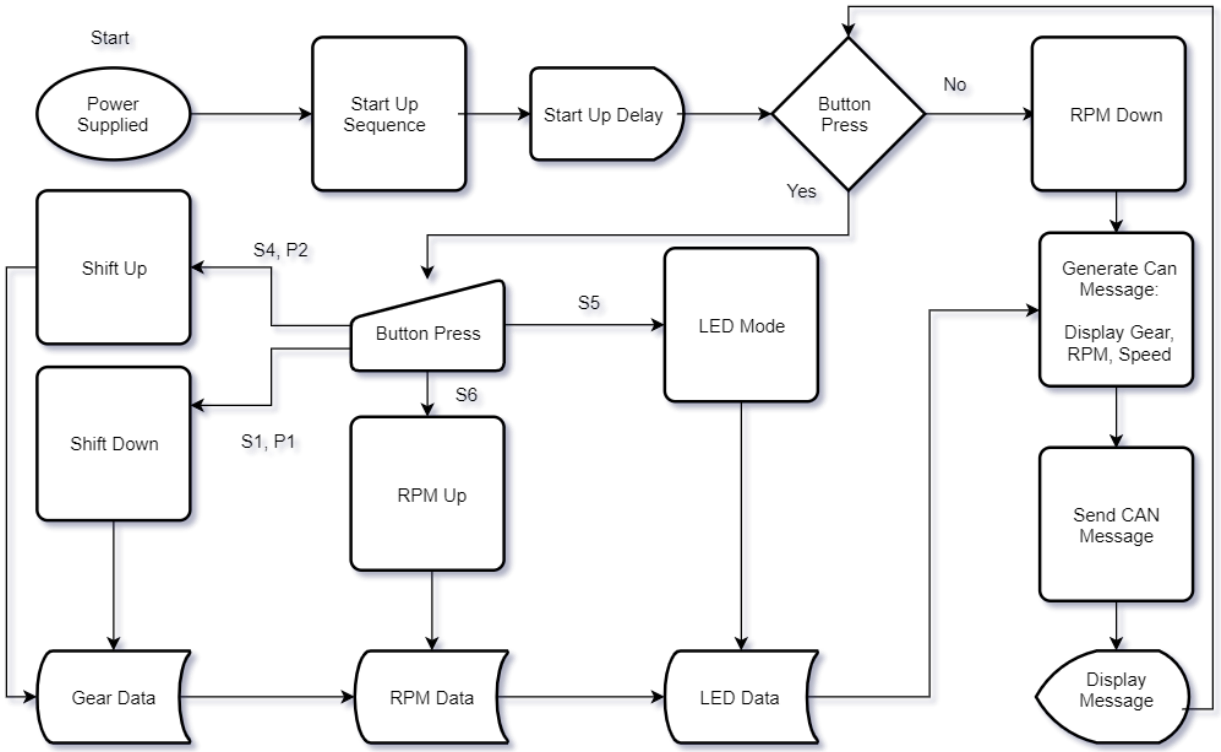


Fig 3- Python code for Steering Wheel and CAN interface

```
import time
import can

#from pylogix import PLC
#comm = PLC()
#comm.Micro800 = True
#comm.IPAddress = '192.168.1.123'#device.IPAddress #change address to match the plc's address
#comm.ConnectionSize = 4000



buttonFilter = [{"can_id":0x00c0, "can_mask": 0x00ff, "extend": False}]
#bus = can.interface.Bus(bustype='socketcan_native', channel = 'can0', can_filters=buttonFilter, bitrate=1000000)
#bus._apply_filters(buttonFilter)
listener = can.Listener()
rpm = 0
rpm2 = 0
pastspeed = 0
speed = 0
speed2 = 0
gear = 1
LedMode = False
flash = False
lastFlash = 0
timestamp = 0
lastTimeStamp = 0
lockOutTimer = 1

leds = can.Message(arbitration_id= 0x1A8, data=[0, 0, 0, 0, 0, 0, 0, 0], extended_id=False)#leds
gears = can.Message(arbitration_id= 0x188, data=[0, 0, 0, gear, 0, 0, 0, 0], extended_id=False)#gear
rpmSpeed = can.Message(arbitration_id= 0x170, data=[0, 0, 0, 0, 0, 0, 0, 0], extended_id=False)#rpm and speed
brightness = can.Message(arbitration_id= 0x72F, data=[0, 0, 0, 0, 0, 0, 0, 0], extended_id=False)#screen brightness

def RPM(rpm):
    global rpm2
    global gear
    global speed
    global speed2

    rpm2 = rpm / 255


    if rpm < 255:
        RPMSpeedCon()
```

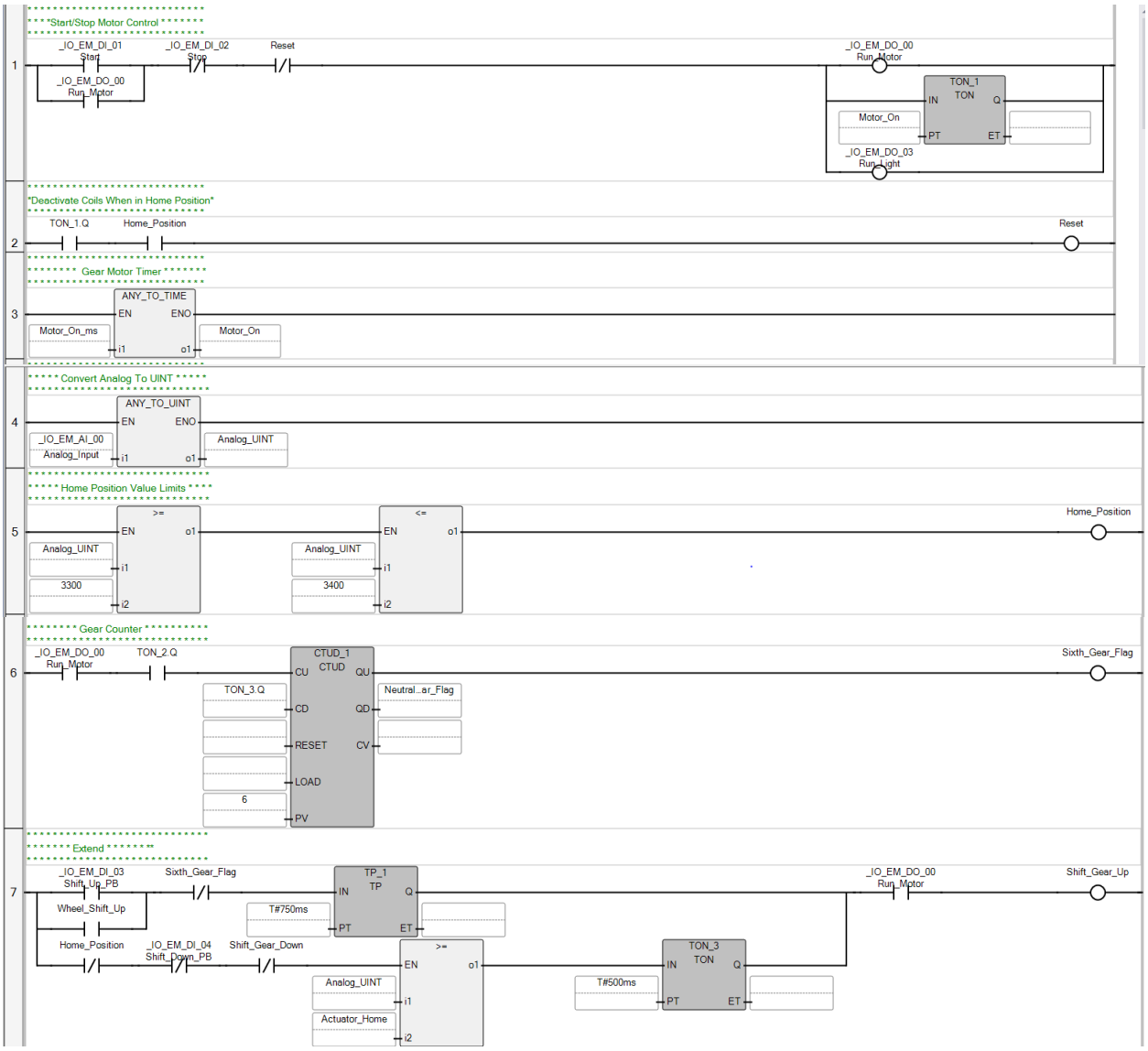
```
45          rpmSpeed.data = [0, int(rpm), 0, 0, 0, 0, 0, int(speed)]
46      else:
47          rpm = rpm - (255 * int(rpm2))
48          RPMSpeedCon()
49          speed = speed - (255* int(speed2))
50          rpmSpeed.data[0] = int(rpm2)
51          rpmSpeed.data[1] = int(rpm)
52          rpmSpeed.data[6] = int(speed2)
53          rpmSpeed.data[7] = int(speed)
54
55  def RPMSpeedCon():
56      global gear
57      global speed
58      global speed2
59      global rpm
60      global rpm2
61      global pastspeed
62
63      if gear == 0x0:  #reverse gear not used but available
64          speed = rpm * .0
65          speed2 = rpm2 * .0
66      elif gear == 0x1: #neutral
67          speed = rpm * .0
68          speed2 = rpm2 * .0
69      elif gear == 0x2: #first gear max 128 mph
70          speed = rpm * .178
71          speed2 = rpm2 * .178
72      elif gear == 0x3: #second gear max 165 mph
73          speed = rpm * .23
74          speed2 = rpm2 * .23
75      elif gear == 0x4: #thired gear max 200 mph
76          speed = rpm * .278
77          speed2 = rpm2 * .278
78      elif gear == 0x5: #fourth gear max 215 mph
79          speed = rpm * .299
80          speed2 = rpm2 * .299
81      elif gear == 0x6: #fifth gear max 225 mph
82          speed = rpm * .313
83          speed2 = rpm2 * .313
84      elif gear == 0x7: #final gear max 233 mph
85          speed = rpm * .324
86          speed2 = rpm2 * .324
```

```
87
88          pastspeed = speed
89
90      def ShiftDown():
91          global gear
92          global timestamp
93          global comm
94          if gear > 1:
95              gear = gear - 1
96          else: gear = 1
97          gears.data = ([0, 0, 0, gear, 0, 0, 0, 0])
98          timestamp = int(message.timestamp)
99          #comm.Write('Wheel_Shift_Down',1) #send signal to plc to shift down
100         #time.sleep(0.1)
101         #comm.Write('Wheel_Shift_Down',0) #turn it off in case it stays on indefinitly
102
103     def ShiftUp():
104         global gear
105         global timestamp
106         global comm
107         if gear < 7:
108             gear = gear + 1
109         else: gear = 7
110         gears.data = ([0, 0, 0, gear, 0, 0, 0, 0])
111         timestamp = int(message.timestamp)
112         #comm.Read('Wheel_Shift_Up')
113         #comm.Write('Wheel_Shift_Up',1) #send signal to plc to shift up
114         #time.sleep(0.1)
115         #comm.Write('Wheel_Shift_Up',0) #turn it off in case it stays on indefinitly
116
117     def LedGear():
118         if gear == 0x0: leds.data[3] = 0x0
119         elif gear == 0x1: leds.data[3] = 0x0
120         elif gear == 0x2: leds.data[3] = 0x1
121         elif gear == 0x3: leds.data[3] = 0x2
122         elif gear == 0x4: leds.data[3] = 0x4
123         elif gear == 0x5: leds.data[3] = 0x8
124         elif gear == 0x6: leds.data[3] = 0x7
125         elif gear == 0x7: leds.data[3] = 0xf
126
127     def LedRPM(rpm):
128         global flash
```

```
    global lastFlash

    if flash == False:
        if rpm < 1500:
            leds.data[3] = 0x0
        elif rpm < 3000:
            leds.data[3] = 0x0 #flash off
        elif rpm < 4500:
            leds.data[3] = 0x1
        elif rpm < 6000:
            leds.data[3] = 0x1 #flash off
        elif rpm < 7500:
            leds.data[3] = 0x3
        elif rpm < 9000:
            leds.data[3] = 0x3 #flash off
        elif rpm < 10500:
            leds.data[3] = 0x7
        elif rpm < 12000:
            leds.data[3] = 0x7 #flash off
        else:
            leds.data[3] = 0xf

        if int(message.timestamp) > (lastFlash +.05):
            flash = not flash
            lastFlash = message.timestamp
    else:
        if rpm < 1500:
            leds.data[3] = 0x0
        elif rpm < 3000:
            leds.data[3] = 0x1 #flash on
        elif rpm < 4500:
            leds.data[3] = 0x1
        elif rpm < 6000:
            leds.data[3] = 0x3 #flash on
        elif rpm < 7500:
            leds.data[3] = 0x3
        elif rpm < 9000:
            leds.data[3] = 0x7 #flash on
        elif rpm < 10500:
            leds.data[3] = 0x7
        elif rpm < 12000:
            leds.data[3] = 0xf #flash on
        else:
            leds.data[3] = 0xf

        if int(message.timestamp) > (lastFlash +.05):
            flash = not flash
            lastFlash = message.timestamp


while True:
    bus = can.interface.Bus(bustype='socketcan_native', channel = 'can0', can_filters=buttonFilter, bitrate=1000000)
    bus.flush_tx_buffer()

    msg = bus._recv_internal(1) # Timeout in seconds.
    message = bus.recv(1)
    #print(devices.IPAddress)
    #GearPLC = comm.Read('CTUD_1.CV') #read the stored value of what gear is stored in the plc

    #print(msg) #renable for debugging infromation of incoming CAN messages.
    #if int(message.timestamp) > (timestamp +lockOutTimer) and (message.data[7] == 0x2):
        #if message.data[7] != lastmessage:


    if message is None:
            print('Timeout occurred, no message.')

    if (message.data[7] == 0x1 or message.data[7] == 0x80) and int(message.timestamp) > (timestamp +lockOutTimer):
        if message.data[7] != lastmessage:
            ShiftDown()
            if gear == 1:
                rpm = rpm  #nuetral gear shift
            elif gear == 2 and rpm > 8000:
                rpm = rpm + 4000  #first gear shift drop 4600
            elif gear == 3 and rpm > 8000:
                rpm = rpm + 3000  #second gear shift drop 9000
            elif gear == 4 and rpm > 8000:
                rpm = rpm + 2000  #third gear shift drop 10000
```

```
            elif gear == 5 and rpm > 8000:
                rpm = rpm + 1000  #fourth gear shift drop 11000
            elif gear == 6 and rpm > 8000:
                rpm = rpm + 750  #fifth gear shift drop 11250
            elif gear == 7 and rpm > 8000:
                rpm = rpm + 500   #final gear shift drop 11500

    if (message.data[7] == 0x8 or message.data[7] ==0x40) and int(message.timestamp) > (timestamp + lockOutTimer):
        if message.data[7] != lastmessage:
            ShiftUp()
            if gear == 1:
                rpm = rpm  #nuetral gear shift
            elif gear == 2 and rpm > 8000:
                rpm = rpm - 4000  #first gear shift drop 4600
            elif gear == 3 and rpm > 8000:
                rpm = rpm - 3000  #second gear shift drop 9000
            elif gear == 4 and rpm > 8000:
                rpm = rpm - 2000  #third gear shift drop 10000
            elif gear == 5 and rpm > 8000:
                rpm = rpm - 1000  #fourth gear shift drop 11000
            elif gear == 6 and rpm > 8000:
                rpm = rpm - 750  #fifth gear shift drop 11250
            elif gear == 7 and rpm > 8000:
                rpm = rpm - 500   #final gear shift drop 11500

    if message.data[7] == 0x020:
        if rpm > 12000:  #Engine max rpm
            rpm = 12000
        else:
            if gear == 0:
                rpm = rpm + 17.3  #reverse gear not being used but available
            elif gear == 1:
                rpm = rpm + 80.3  #nuetral gear
            elif gear == 2:
                rpm = rpm + 70.7  #first gear
            elif gear == 3:
                rpm = rpm + 60.1  #second gear
            elif gear == 4:
                rpm = rpm + 50.5  #third gear
            elif gear == 5:
                rpm = rpm + 40.2  #fourth gear
            elif gear == 6:
                rpm = rpm + 30.9  #fifth gear
            elif gear == 7:
                rpm = rpm + 20.6   #final gear
    elif rpm > 45:
        rpm = rpm - 20.6
    else:
        rpm = 0


    if message.data[7] == 0x10:
        if message.data[7] != lastmessage:
            LedMode = not LedMode



    if LedMode == False:
        LedGear()
    elif LedMode == True:
        LedRPM(rpm)
    else:
        LedGear()

    RPM(rpm)

    bus.send(rpmSpeed)
    time.sleep(.01)
    bus.send(gears)
    time.sleep(.01)
    bus.send(leds)
    time.sleep(.01)

    lastmessage = message.data[7]
    bus.shutdown()
    #time.sleep(.05)
```

Figure 4- Connected Components Workbench Software (CCW)



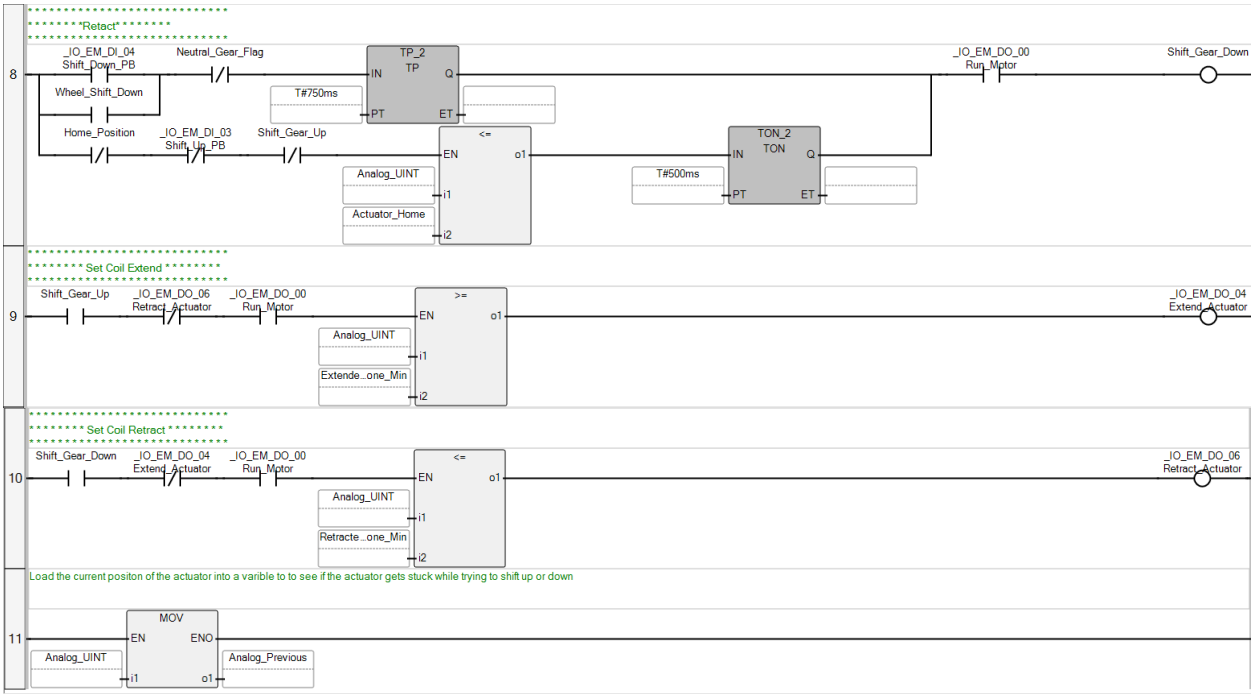
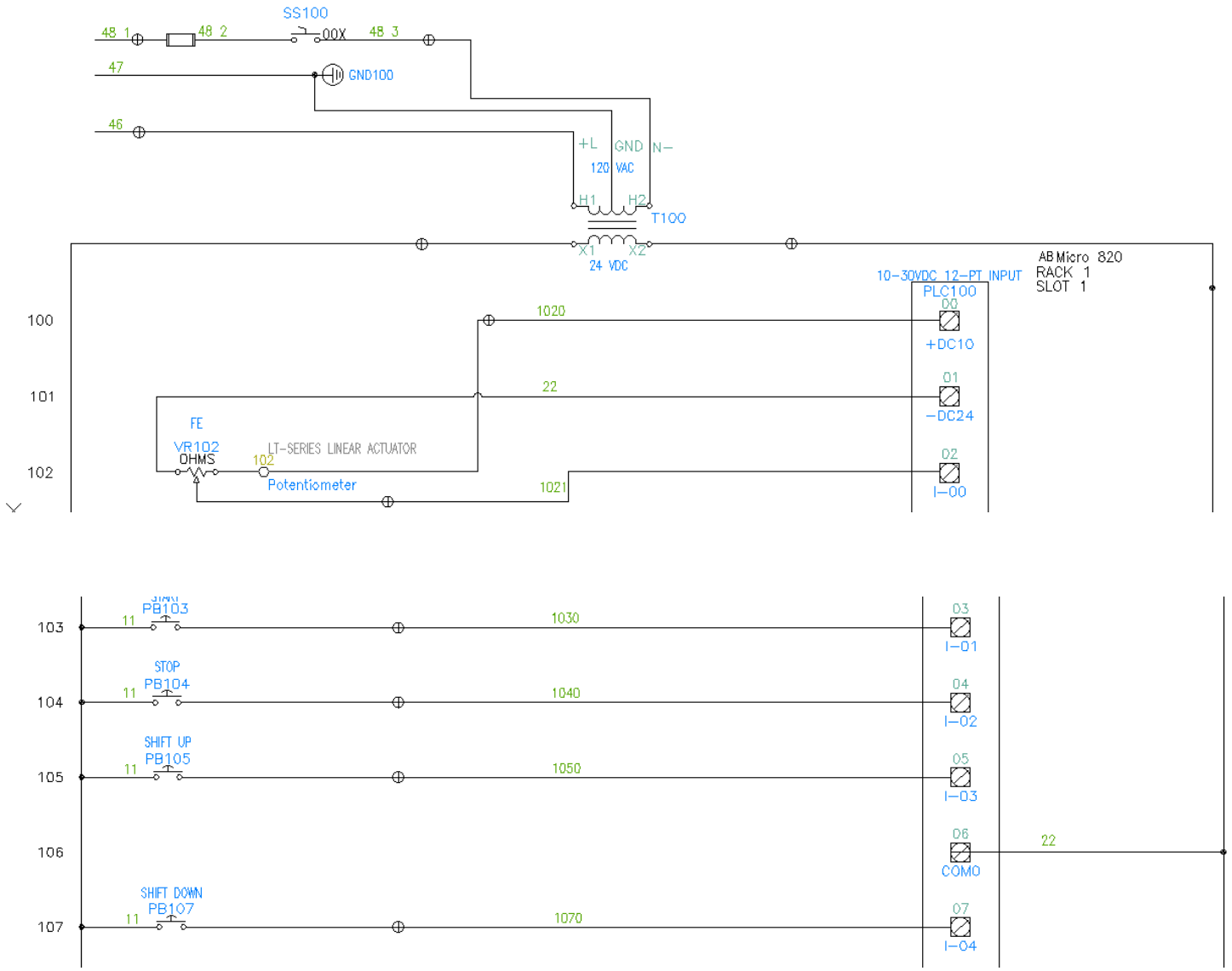
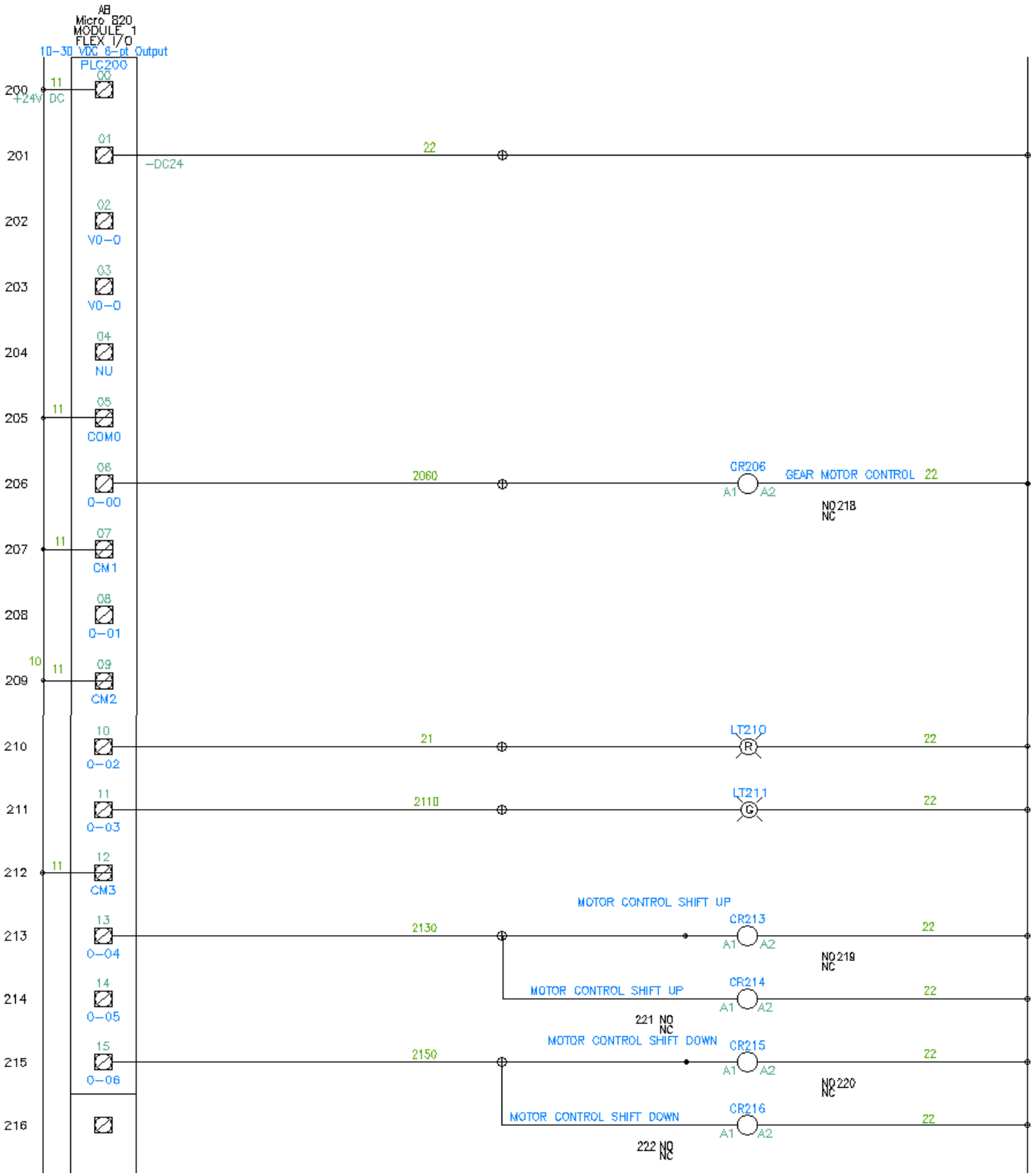


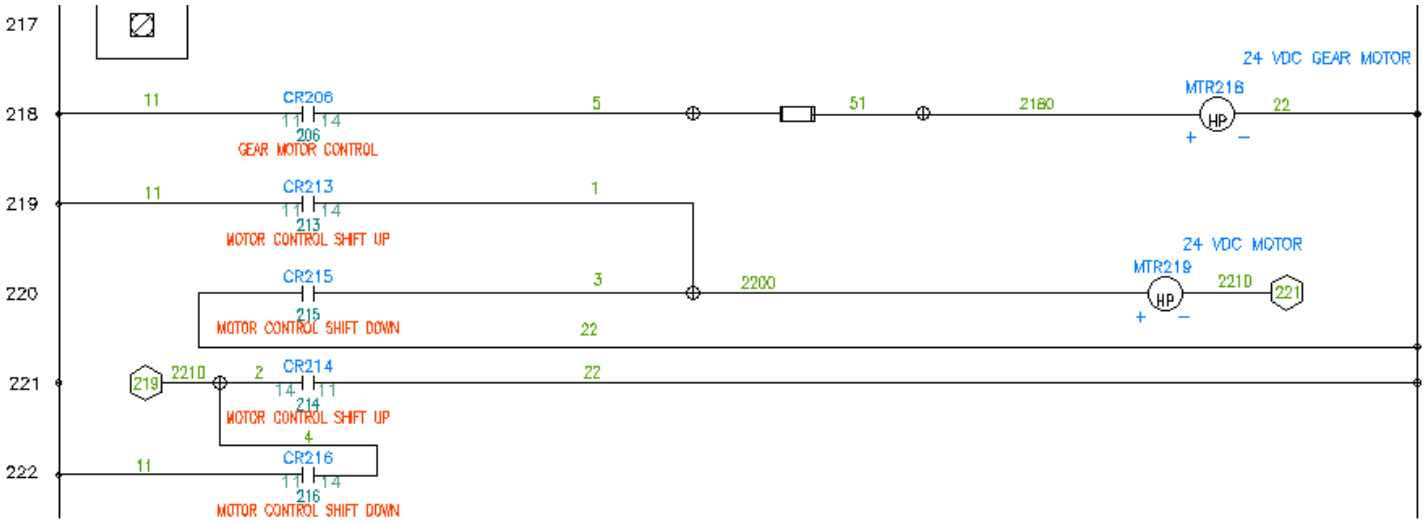
Figure 5- PLC Input/Output

Name	Alias	Data Type	Dimension	Project Val	Initial Value	Comment	Retained	String Size
_IO_EM_DO_00	Run_Motor	BOOL						
_IO_EM_DO_01		BOOL						
_IO_EM_DO_02	Stop_Light	BOOL						
_IO_EM_DO_03	Run_Light	BOOL						
_IO_EM_DO_04	Extend_Actuator	BOOL				Power the actuator so that extends		
_IO_EM_DO_05		BOOL						
_IO_EM_DO_06	Retract_Actuator	BOOL				Power the actuator so that it retracts		
_IO_EM_DI_00		BOOL						
_IO_EM_DI_01	Start	BOOL						
_IO_EM_DI_02	Stop	BOOL						
_IO_EM_DI_03	Shift_Up_PB	BOOL						
_IO_EM_DI_04	Shift_Down_PB	BOOL						
_IO_EM_DI_05	Actuator_Home_LS	BOOL						
_IO_EM_DI_06	Actuator_Extended_	BOOL						
_IO_EM_DI_07	Actuator_Retracted	BOOL						
_IO_EM_DI_08		BOOL						
_IO_EM_DI_09		BOOL						
_IO_EM_DI_10		BOOL						
_IO_EM_DI_11		BOOL						
_IO_EM_AI_00	Analog_Input	WORD						
_IO_EM_AI_01		WORD						
_IO_EM_AI_02		WORD						
_IO_EM_AI_03		WORD						
_IO_EM_AO_00		WORD						
Motor_On		TIME						
Analog_UINT		UINT				The raw input signal from the linear actuator		
ShiftDown_Latch		BOOL				Hold timer while shifting down		
ShiftUpMax		DINT			5		✓	
ShiftUp_Latch		BOOL						
Analog_Previous		UINT				Previous position the actuator was in		

Figure 6: Control Box Schematic – Power and Inputs



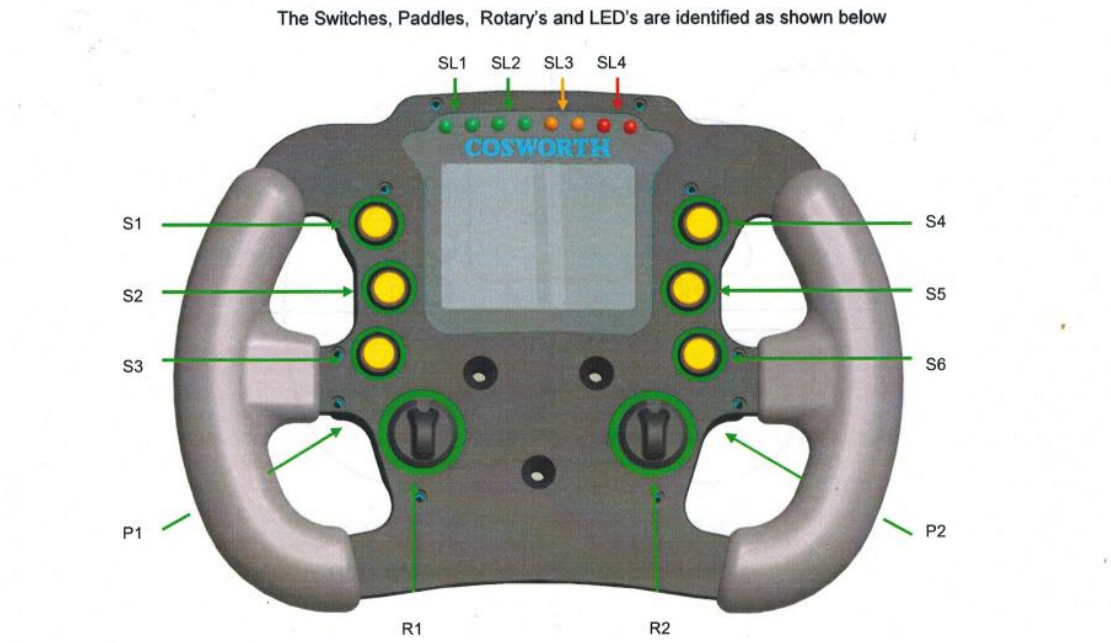

AB
Micro 820
MODULE 1
FLEX I/O
10-30 VDC 6-pt Output
PLC200
200
+24V DC
11
00
201
01
22
-DC24
202
02
V0-0
203
03
V0-0
204
04
NU
205
11
05
COM0
206
06
2060
0-00
CR206
GEAR MOTOR CONTROL 22
A1 A2
NO 218
NC
207
11
07
CM1
208
08
0-01
209
10
11
09
CM2
210
10
21
0-02
LT210
R
22
211
11
2110
0-03
LT211
G
22
212
11
12
CM3
MOTOR CONTROL SHIFT UP
213
13
2130
0-04
CR213
A1 A2
22
NO 219
NC
214
14
0-05
MOTOR CONTROL SHIFT UP
CR214
A1 A2
22
221 NO
NC
MOTOR CONTROL SHIFT DOWN
215
15
2150
0-06
CR215
A1 A2
22
NO 220
NC
216
MOTOR CONTROL SHIFT DOWN
CR216
A1 A2
22
222 NO
NC


217
218
219
220
221
222
24 VDC GEAR MOTOR
MTR218
HP
11
CR206
206
GEAR MOTOR CONTROL
5
51
2180
22
CR213
213
MOTOR CONTROL SHIFT UP
1
24 VDC MOTOR
MTR219
2210
221
CR215
215
MOTOR CONTROL SHIFT DOWN
3
2200
22
219
CR214
214
MOTOR CONTROL SHIFT UP
2
4
CR216
216
MOTOR CONTROL SHIFT DOWN

Interface Design

Below showcases the interface of the steering wheel with the respective inputs and outputs per the schematic.

Figure 7 – Schematic of Steering Wheel Display



The steering wheel has eight separate digital switch inputs (6 buttons on the front and 2 paddles on the back) as displayed in Figure 1:

- Both **Button 1 (S1)** and **Paddle 1 (P1)** will shift the gears down
- **Button 2 (S2)** starts/stops the motor
- **Button 3 (S3)** changes pages on the display
- Both **Button 4 (S4)** and **Paddle 2 (P2)** shifts the gear up
- **Button 5 (S5)** LEDs from RPM to gear
- **Button 6 (S6)** simulate acceleration.

There are 8 shift light LED's at the top of the wheel which configure into 4 pairs (SL1-SL4)

- **LED 1 and 2 GREEN (SL1)** represents when gears are in first shift
- **LED 3 and 4 GREEN (SL2))** represents when gears are in second shift
- **LED 5 and 6 ORANGE (SL3))** represents when gears are in third shift
- **LED 7 and 8 ORANGE (SL4))** represents when gears are in fourth shift
- When SL1 and SL2 are on, the gears are in fifth shift
- When SL3 and Sl4 are on, the gears are in sixth shift
- When all LEDs are on, the gears are in neutral shift.

Low Level Design

Hardware

As displayed on the Board Selection Matrix below, the hardware interface was chosen based on the factors of price, built-in CAN interface, CAN peripheral, library supported CAN chip, operating environment robustness, and the time it takes to implement code. The Raspberry Pi was chosen over the other options due to its cost-effectiveness.

Table 3 – Board Selection Matrix

Board Selection Matrix							
Choices	Price	Built in Can Interface	Can Peripheral	Library Supported CAN Chip	Operating Environment Robustness	Time to implement	Total Criteria
Raspberry Pi 3+	1 $38.10	0	1 PiCAN2 - $49.95	1 MCP2515	0.5	1	4.5
Pine A64	1 $32.00	0	0	0	0.5	0	1.5
Arduino Uno Rev3	1 $22.00	0	1 CAN Shield - $33.90	1 MCP2515	0.5	0.5	4
Zebra VL-EPC-2701-EAK-005	0.5 $196.00	1	1	0.5 Custom API by Manufacturer	1	0.5	4.5
Commission Custom Board	0 Quoted Price	1	1	1	1	0	4

Table 4 – Power Supply Selection Matrix

Power Supply Selection Matrix						
Choices	Price	Compatible Connectivity	Designed to Specifications	IEC Safety Standards	Time to implement	Total Criteria
Custom Built Power Supply	0.5	1	1	0.5	0.5	3.5
Pre-Built Power Supply	1	0.5	0.5	0.5	1	3.5
Commissioned Power Supply	0	1	1	1	0	3

Table 5 – Coding Language Selection Matrix

Coding Language Selection Matrix							
Choices	CAN Library Availability	Platform Restrictions	Level of abstraction	Strongly Typed	Speed	Time to implement	Total Criteria
C#	1	1 Windows IOT	0.5 High	1	0.5	0.5	4.5
C++	1	1 Windows IOT Raspbian	1 Low	1	1	0.5	5.5
C	1	0.5 Raspbian	1 Low	1	1	1	5.5
Java	1	0.5 Linux Raspbian	0.5 High	1	0.5	0.5	4

Python	1	0.5 Linux Raspbian	0.5 High	0	0	0.5	2.5

Interface

Below is the low-level design interface in which the PLC will interface with a Raspberry Pi through an Ethernet cable. Through PICAN2, it will communicate to the CAN formula wheel.

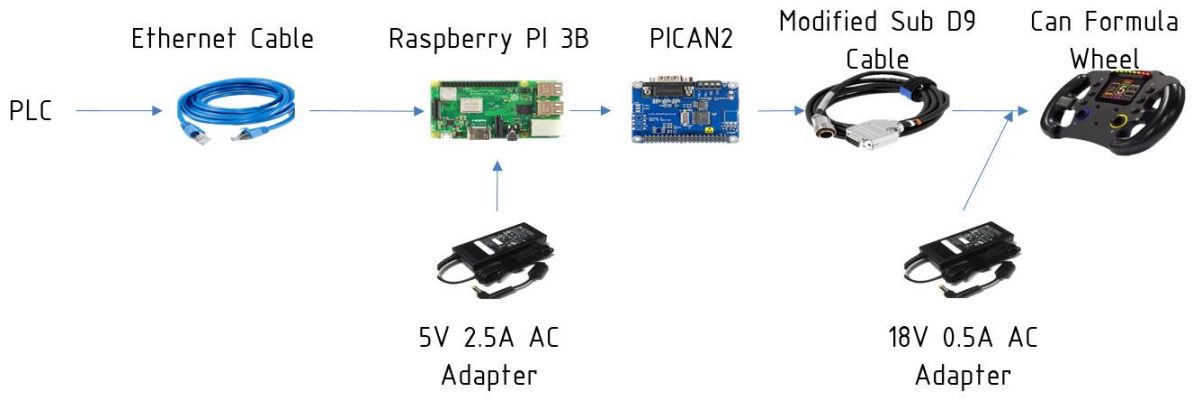


Figure 8: Original Interface

However, upon the new release of the PI-Hat, a more energy efficient and cost effective interface was devised. Instead of utilizing the PICAN2, which required an additional AC adapter and a modified sub cable, the Raspberry PI will connect with the PI-Hat and interface with the CAN formula wheel.

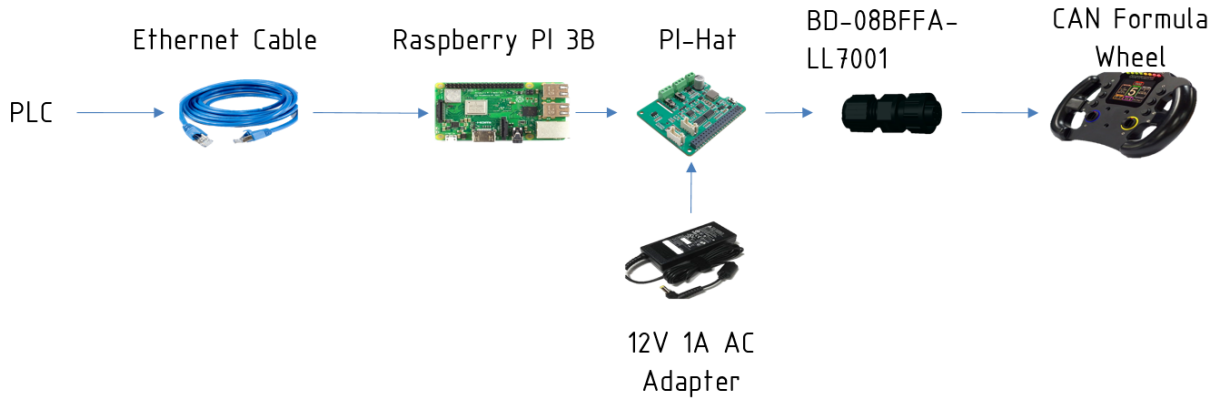


Figure 9: Current Interface

Setup and Operation

Before we begin, check the metal tray beneath the transmission display. There should be a red rotary switch mounted on the side.

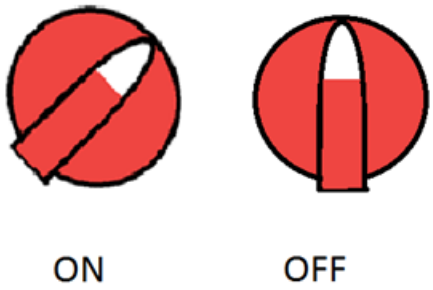


Figure 10: Switch On/Off

Make sure that the switch is in an upright position so that the system is off (as displayed in Figure 1).

The first step is to plug in the transmission's power supply into an electrical outlet.

Once the power supply has been secured into position, turn the system on by turning the rotary switch to the right (as displayed in Figure 1).

There should be a 1-2 minute delay as the steering wheel attached to the display turns on.

Once the steering wheel displays is on (as seen in Figure 2), the simulation is ready for launch:

Figure 11: Steering Wheel Display

Qualification Testing

Market Req.	Engineering Req.	Compliance Determined	Accept/Reject Criteria	Test Required
Interface wheel with control box	Secure connection between steering wheel and control box	Adheres to CAN communication SAE J1939 and IEEE 29119-1-2013	Pass the requirements of SAE J1939 and IEEE 29119-1-2013 **Pass**	Yes SAE J1939 and IEEE 29119-1-2013
Change gears through display and paddles	Shift gears using the paddles on the wheel	Adheres to IEEE 15026-1-2014 and ISA108	Pass the requirements of IEEE 15026-1-2014 and ISA108 **Pass**	Yes IEEE 15026-1-2014 and ISA108
Change gears through display and paddles	Shift gears using on screen controls	Adheres to IEEE 15026-1-2014 and ISA108	Pass the requirements of IEEE 15026-1-2014 and ISA108 **Pass**	Yes IEEE 15026-1-2014 and ISA108
LED indicators	LEDs reflects the current gear shifted	Adheres to IEEE 29119-1-2013	Pass the requirements of IEEE 29119-1-2013 **Pass**	Yes IEEE 29119-1-2013
LED indicators, Display RPM	LEDs reflect current RPM of engine	Adheres to IEEE 29119-1-2013	Pass the requirements of IEEE 29119-1-2013 **Pass**	Yes IEEE 29119-1-2013
Display which gears are shifted	Screen reflects current gear shifted	Adheres to IEEE 29119-1-2013	Pass the requirements of IEEE 29119-1-2013 **Pass**	Yes IEEE 29119-1-2013
Interface wheel with control box	Screen reflects approximate speed	Adheres to IEEE 29119-1-2013	Pass the requirements of	Yes IEEE 29119-1-2013

			IEEE 29119-1-2013 **Pass**	
Interface wheel with control box	Push Button 1 simulates acceleration	Adheres to IEEE 29119-1-2013	Pass the requirements of IEEE 29119-1-2013 **Pass**	Yes IEEE 29119-1-2013
Interface wheel with control box	Push button 3 changes MPH to KPH on display	Adheres to IEEE 29119-1-2013	Pass the requirements of IEEE 29119-1-2013 **Pass**	Yes IEEE 29119-1-2013
Interface wheel with control box, Display RPM, LED indicators	Push Button 2 shift LEDs from RPM to gear	Adheres to IEEE 29119-1-2013	Pass the requirements of IEEE 29119-1-2013 **Pass**	Yes IEEE 29119-1-2013
Interface wheel with control box	Secure connection from microcontrollers to PLC	Adheres to IEEE 15026-1-2014 and ISA108	Pass the requirements of IEEE 15026-1-2014 and ISA108 **Pass**	Yes IEEE 15026-1-2014 and ISA108

Test Plans

Check Power to Steering Wheel

Purpose	Based on the specifications for CAN communications (SAE J1939) and software testing (IEEE 29119-1-2013), the purpose of this test is to ensure that the system, when activated will communicate with the computer and the program.
Location	ET 2
Materials	PI-CAN, Computer

Step	Actions	Expected Results	Observed Results
1.	Turn on system power.	Verify System power.	When plugging into the outlet, the wheel takes 1-2 minutes to turn on
2.	Initialize sequence	Verify sequence.	Splash screen appears.
3.	Panel View Screen Initialization sequence starts.	Verify Panel View Display Initializes.	Initialization screen appears
4.	Communications established to Laptop computer	Communications established.	The code has become
5.	Reset all system alarms	System alarms cleared	System alarms cleared
6.	Turn off system power.	Verify process skid power shuts down PLC, Panel View Display and communications with laptop is lost.	Process verified.
7.	Restore System Power.	Power is Restored	Power restored.

Test State: X Passed ____ Failed Test Run #_____

Tested by: Cristobal Ibanez Date:_____

Comments:

Checking Button

Purpose	Based on the specifications for CAN communications (SAE J1939), software testing (IEEE 29119-1-2013) and PLC involvement (IEC 1131-3), the purpose of this test is to ensure that the steering wheel and the control box send signals back and forth to one another.
Location	ET 2
Materials	Control Box, PI-CAN, Computer

Step	Actions	Expected Results	Observed Results
1.	Press Button 1	Shift gear down	The screen shows that the simulation has shifted down
2.	Press Button 2	Acceleration	The acceleration box shows the numbers increasing the longer the button is held
3.	Press Button 3	Shift LED from RPM to gear display	The display changes when button is pressed
4.	Press Button 4	Shift Gear Up	The screen shows that the simulation has shifted up
5.	Press Button 5	Stop	Confirmed stop. Must test for integration
6.	Press Button 6	Change display from KPH to MPH	Display changes from KPH to MPH and vice versa
7.	Press Paddle 1	Shift gear down	The screen shows that the simulation has shifted down
8.	Press Paddle 2	Shift Gear Up	The screen shows that the simulation has shifted up

Test State: X Passed ____ Failed Test Run #_____

Tested by: Cristobal Ibanez Date:_____

Comments:

Checking Actuator

Purpose	Following **IEC 1131-3,** this test is to ensure that the actuator will extend and retract correctly and separately from the transmission, thereby influencing the movement of the ratchet and shift up and down the gears.
Location	ET 220
Materials	Gearbox, computer, Ethernet cable, micro820 Allen Bradley programmable controls

Step	Actions	Expected Results	Observed Results
1.	When “start” activates	The system turns on	There is a click in the PLC that indicates the system has turned on.
2.	When “stop” activates	The system turns off	There is a click in the PLC that indicates the system has turned on. Testing the other wires reveals that the act was successful
3.	When “shift up” activates	Actuator extends before returning to home	The actuator extends then returns to home. However, when the shift up/down activates whilst the actuator is returning to home, the process is interrupted and continues with the new process
4.	When “shift down” activates	Actuator retract before returning to home	The actuator extends then returns to home. However, when the shift up/down activates whilst the actuator is returning to home, the process is interrupted and continues with the new process

Test State: X Passed ____ Failed Test Run #_____

Tested by: Charleston Shi and Cristobal Ibanez Date:_____

Comments:

Checking Actuator whilst in Ratchet

Purpose	Following **IEC 1131-3,** this test is to ensure that the actuator will extend and retract correctly with the transmission, thereby influencing the movement of the ratchet and shift up and down the gears.
Location	ET 220
Materials	Gearbox, computer, Ethernet cable, micro820 Allen Bradley programmable controls

Step	Actions	Expected Results	Observed Results
1.	When "start" activates	The system turns on	There is a click in the PLC that indicates the system has turned on.
2.	When "stop" activates	The system turns off	There is a click in the PLC that indicates the system has turned on. Testing the other wires reveals that the act was successful
3.	When "shift up" activates	Actuator extends before returning to home. Gears shift up.	Pre-Fix: The actuator varies in results. Sometimes it extends and nothing happens. Sometimes it extends and the gears do shift. Post-Fix: After replacing the actuator's plunger and fixing the actuator's internal wirings, the gears successfully shift up
4.	When "shift down" activates	Actuator retract before returning to home. Gears shift down.	Pre-Fix: The actuator does not shift the gears down. Post-Fix: After replacing the actuator's plunger and fixing the actuator's internal wirings, the gears successfully shift down

Test State: X Passed ____ Failed Test Run #_____

Tested by: Charleston Shi and Cristobal Ibanez Date:_____

Comments:

Checking Shift Up/Down While Motor is implemented

Purpose	Following **IEC 1131-3,** this test is to ensure that the actuator will extend and retract correctly with the transmission when the motor has been implemented, thereby influencing the movement of the ratchet and shift up and down the gears.
Location	ET 220
Materials	Gearbox, computer, Ethernet cable, micro820 Allen Bradley programmable controls

Step	Actions	Expected Results	Observed Results
1.	When "start" activates	The system turns on	There is a click in the PLC that indicates the system has turned on.
2.	When "stop" activates	The system turns off	There is a click in the PLC that indicates the system has turned on. Testing the other wires reveals that the act was successful
3.	Motor load test	The motor spins when wires are connected	The motor spins while wires are connected
4.	Motor implementation	The motor spins the gears	The motor spins the gears
5.	When "shift up" activates	Actuator extends before returning to home. Gears shift up.	Pre-Fix: The actuator varies in results. Sometimes it extends and nothing happens. Sometimes it extends and the gears do shift. Post-Fix: After replacing the actuator's plunger and fixing the actuator's internal wirings, the gears successfully shift up
6.	When "shift down" activates	Actuator retract before returning to home. Gears shift down.	Pre-Fix: The actuator does not shift the gears down. Post-Fix: After replacing the actuator's plunger and fixing the actuator's internal wirings, the gears successfully shift down

Test State: X Passed ____ Failed Test Run #_____

Tested by: Charleston Shi and Cristobal Ibanez Date:_____

Implementation of Steering Wheel

Purpose	Following **IEC 1131-3,** this test is to ensure that the actuator will extend and retract correctly with the transmission when the motor has been implemented, thereby influencing the movement of the ratchet and shift up and down the gears.
Location	ET 220
Materials	Gearbox, computer, Ethernet cable, micro820 Allen Bradley programmable controls

Step	Actions	Expected Results	Observed Results
1.	Press Button 1	Shift gear down	TBD
2.	Press Button 2	Acceleration	TBD
3.	Press Button 3	Shift LED from RPM to gear display	TBD
4.	Press Button 4	Shift Gear Up	TBD
5.	Press Button 5	Stop	TBD
6.	Press Button 6	Change display from KPH to MPH	TBD

Comments: **As of this writing, the integration has not been tested and is still in the implementation process. Cristobal and Charleston are currently working with this section.**

Conclusion

Through his project IMS is allowing engineering students the opportunity to gain experience and be involved with the racing community as well as have experience in a high-profile project. As such, the purpose of this to interface a racecar steering wheel with engine transmission box in order to send signals back and forth upon activation.

Recommendations

In the event that the interface will not be complete by December 20, 2019, the necessary actions need to be taken are as such per the current and/or future specifications of the sponsor:

- The interface is the communication of the steering wheel to the Micro 820 Controller. All controls will have to be transferred to the steering wheel and must be able to directly communicate with the current program downloaded to the controller

References

Table 6 – Standards and Related Documents

Title	Source	Comment
IEEE 15026-1-2014	https://standards.ieee.org/standard/15026-1-2014.html	Potential code case structures and contents
IEC 1131-3	http://isa.uniovi.es/genia/english/publicaciones/IEC%20%201131-3.pdf	Possible PLC involvement in the process
ISO/IEC/IEEE 29119-1-2013	http://www.softwaretestingstandard.org/	Software testing
ISA Standards	https://www.isa.org/standards-publications/	Recognized automation standards
NFPA 79-2018	https://www.nfpa.org/codes-and-standards/all-codes-and-standards/list-of-codes-and-standards/detail?code=79	provides safeguards for industrial machinery to protect operators, equipment, facilities, and work-in-progress from fire and electrical hazards
29D-032985-U	www.cosworth.com	Wheel Specifications and diagrams